UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Geociências e Ciências Exatas - IGCE
Curso de Bacharelado em Ciências da Computação

MAICON DALL'AGNOL

TRABALHO DE CLASSIFICAÇÃO

Professora: Dra. Adriane Beatriz de Souza Serapião

Rio Claro - SP

2019

1 Introdução

Este trabalho consiste em aplicar o conhecimento de Classificadores adquirido na disciplina Tópicos: Aprendizado de Máquina. Tem-se como objetivo:

- Escolha dois datasets rotulados.
- Realize a análise estatística, visualização e pré-processamento dos dados.
- Realize os experimentos criando duas bases de teste distintas:
  - considerando todos os atributos do dataset;
  - selecionando alguns atributos e descartando outros.
- Aplique três métodos de classificação distintos nas duas bases acima referentes a cada dataset.
- Para cada dataset, em cada uma das bases, analise os resultados segundo medidas de qualidade de classificação, usando índices de validação externa (acurácia, recall, precisão, F-measure, índice Kappa) e cruva ROC.
- Proponha uma maneira adicional de comparar os resultados obtidos além das medidas acima.
- Compare e interprete os resultados dos dois experimentos em cada dataset.
- Faça tabela com as medidas de validação

2 Desenvolvimento

Inicialmente foram escolhidos dois *datasets* com diferentes aspectos. O primeiro a ser abordado refere-se a riscos de crédito de 1000 instancias (DUA; GRAFF, 2017), onde cada uma contem dados como status da conta corrente, histórico de crédito, objetivo do crédito, quantidade de crédito, entre outros atributos (20 atributos no total) e um atributo alvo, o risco de crédito: bom, equivalente a 0, e ruim, equivalente a 1.

O segundo dataset corresponde a fonemas e é composto de atributos abstratos cujo atributo alvo é a classe de som nasais ou orais, para o dataset em questão classe 1 e 2, respectivamente.

2.1 Pré-processamento e Visualização

Para ambos *datasets* utilizou-se de um biblioteca do Python chamada Pandas Profiling que descreve e faz uma pré análise dos *datasets*, como linhas duplicadas, dados faltantes, grande variância entre os dados.

A partir desta análise notou-se que em ambos datasets o conjunto de classes é desbalanceado, entre elas de forma que em média 1/3 pertence a uma classe e 2/3 a outra. Para o *dataset* de crédito optou-se por utiliza-lo no formato que carregado, enquanto para o dos fonemas realizou-se um rebalanceamento das classes por *under-sample*.

Para o *dataset* de crédito notou-se que os dados em ambas classes estão bem distribuídos entre os atributos de modo que não há um atributo que separe os dados de forma linear. No *dataset* de fonemas há um ligeira separação dos dados podendo ser visível ao plotar o atributo V1 contra os outros. Visualizando os correlogramas não notou-se nenhum atributo com alta correlação com as classes.

Antes da aplicação dos algoritmos de classificação para o *dataset* de crédito necessitou-se de uma transformação, uma vez que os valores de certos atributos correspondem a dados categóricos, portanto para estes aplicou-se um algoritmo chamado *LabelEncoder* que transformou dados categóricos em valores discretos. Dado a transformação para valores numéricos, utilizou-se de um algoritmo escalonador(*StandardScaler*), deixando todos atributos num alcance igualitário. Estas transformações não foram necessárias para o fonemas uma vez que o dataset já se encontrava em formato numérico e escalonado.

2.2 Classificação

Para classificação dos algoritmos tomou-se como método de avaliação o *K-Fold Cross Validation* com K=10. Utilizando então o K-Fold tomou-se 4 algoritmos diferentes KNN, GaussianNB, DecisionTreeClassifier e SVM, todos estes implementados pela biblioteca Scikit-learn.

Para o dataset de crédito aplicou-se os algoritmos no dataset completo e com certos atributos, em ambos o dataset foi escalonado. Para aplicação dos algoritmos nos fonemas optou-se por 3 métodos, primeiro utilizando o conjunto de dados completo sem nenhum balanceamento, utilizando o conjunto dados com redução espacia (no caso utilizou-se PCA) e o conjunto de dados balanceado.

2.3 Avaliação

Para avaliar os resultados obtidos pelos algoritmos tomou-se a média das k-execuções dos algoritmos. Os resultados são apresentados na Tabela 1 e Tabela 2, em que as cédulas verdes correspondem aos maiores valores para a medida de de avaliação entre todos as execuções dos *datasets* e as cédulas vermelhas, as piores.

Cru	Acurácia	Recall	Precisão	F1	Roc	Kappa	Acurácia Balanceada
KNN	0.6670	0.7861	0.7485	0.7661	0.5876	0.1792	0.5876
Gauss	0.6890	0.7619	0.7950	0.7730	0.6488	0.2805	0.6488
Tree	0.6950	0.7719	0.7855	0.7771	0.6412	0.2811	0.6412
SVM	0.7460	0.9375	0.7573	0.8370	0.6182	0.2780	0.6182
Selecionado							
KNN	0.6280	0.7341	0.7352	0.7334	0.5578	0.1127	0.5578
Gauss	0.7040	0.9404	0.7220	0.8161	0.5476	0.1180	0.5476
Tree	0.6080	0.7045	0.7254	0.7139	0.5426	0.0833	0.5426
SVM	0.6940	0.9506	0.7111	0.8128	0.5238	0.0586	0.5238

Tabela 1 – Resultados *dataset* de crédito.

Para o *dataset* de crédito, Tabela 1, é nítido que os melhores resultados foram obtidos pelo *dataset* completo, pois 6 das 7 melhores medidas de avaliação estão nele, enquanto todos os menores valores estão no *dataset* com atributos selecionados. Entre os algoritmos, destaca-se GaussianNB e SVM, com 3 e 2, respectivamente, dos maiores valores obtidos. Ainda neste *dataset* é possivel notar a diferença entre a Acurácia e Acurácia Balanceada, em todos os casos esta segunda obteve valores mais baixos que a primeira, isto ocorre pois como há um desbalanço de classe os resultados tendem a ficar "maquiados", contudo a acurácia balanceada tende a diminuir isto, mostrando um valor de acurácia mais realista.

Cru	Acurácia	Recall	Precisão	F1	Roc	Kappa	Acurácia Balanceada
KNN	0.9032	0.9449	0.9204	0.9324	0.8734	0.7612	0.8734
Gauss	0.7602	0.7735	0.8731	0.82	0.7502	0.4636	0.7502
Tree	0.8712	0.9108	0.9072	0.9090	0.8432	0.6879	0.8432
SVM	0.8487	0.8964	0.8903	0.8933	0.8148	0.6327	0.8148
Balanceado							
KNN	0.8701	0.8684	0.8721	0.8699	0.8702	0.7395	0.8702
Gauss	0.7442	0.6632	0.7925	0.7212	0.7440	0.4875	0.7440
Tree	0.8365	0.8388	0.8372	0.8374	0.8363	0.6722	0.8363
SVM	0.8438	0.7711	0.9033	0.8310	0.8440	0.6871	0.8440
PCA							
KNN	0.7802	0.8450	0.8441	0.8445	0.7338	0.4679	0.7338
Gauss	0.7792	0.7870	0.8881	0.8344	0.7732	0.5058	0.7732
Tree	0.7755	0.8364	0.8449	0.8405	0.7318	0.4606	0.7318
SVM	0.7952	0.8322	0.8723	0.8517	0.7683	0.5199	0.7683

Tabela 2 – Resultados *dataset* de fonemas.

Para o *dataset* de fonemas, Tabela 2, o algoritmo KNN aplicado nos dados sem nenhum balanceamento ou redução espacial obteve os melhores resultados em todas as medidas, enquanto os piores ficaram divididos entre GaussianNB aplicado nos dados balanceados e DecisionTree aplicado nos dados com a transformação do PCA. Neste caso é possivel notar ainda que os resultados da acurácia e acurácia balanceada no caso em que o *dataset* está balanceado, a diferença entre elas é ínfima, diferente dos outros dados que variam entre $0,5\%$ à 5%

Classificacao

May 21, 2019

1 0. Introdução

Trabalho:
Aluno: Maicon Dall'Agnol
R.A.: 151161868
Disciplina: Tópico em Aprendizado de Máquina
Objetivos :

- Escolha dois datasets rotulados.
- Realize a análise estatística, visualização e pré-processamento dos dados.
- Realize os experimentos criando duas bases de teste distintas:
- – considerando todos os atributos do dataset;
- – selecionando alguns atributos e descartando outros.
- Aplique três métodos de classificação distintos nas duas bases acima referentes a cada dataset.
- Para cada dataset, em cada uma das bases, analise os resultados segundo medidas de qualidade de classificação, usando índices de validação externa (acurácia, recall, precisão, F-measure, índice Kappa) e cruva ROC.
- Proponha uma maneira adicional de comparar os resultados obtidos além das medidas acima.
- Compare e interprete os resultados dos dois experimentos em cada dataset.
- Faça tabela com as medidas de validação

1.1 0.1 Dependências

Para realização da tarefa foram utilizados as seguintes bibliotecas:

```
In [2]: import pandas as pd
        import numpy as np
        import pandas_profiling

        from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.preprocessing import StandardScaler

# KFold
from sklearn.model_selection import KFold
import random

# Classificadores
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

#Metricas
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from  sklearn.metrics  import  roc_auc_score
from sklearn.metrics import balanced_accuracy_score

#Visualização
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

2 1. Dados

Este conjunto de dados classifica as pessoas descritas por um conjunto de atributos como riscos de crédito bons ou ruins

2.1 1.1 Informações sobre os dados:

Atributos:

- Status da conta corrente existente, no marco alemão.
- Duração em meses
- Histórico de crédito (créditos recebidos, devolução devida, atrasos, contas críticas)
- Objetivo do crédito (carro, televisão, . . .)
- Quantidade de crédito
- Situação da conta-poupança / títulos, no marco alemão.
- Emprego atual, em número de anos.
- Taxa de parcelamento em percentagem do rendimento disponível

- Estatuto pessoal (casado, solteiro, ...) e sexo
- Outros devedores / fiadores
- Residência atual desde X anos
- Propriedade (por exemplo, imóveis)
- Idade em anos
- Outros planos de parcelamento (bancos, lojas)
- Habitação (aluguel, próprio, ...)
- Número de créditos existentes neste banco
- Trabalho
- Número de pessoas susceptíveis de fornecer manutenção para
- Telefone (sim, não)
- trabalhador estrangeiro (sim, não)

Classe:

- Class

2.2 Importando Dataset

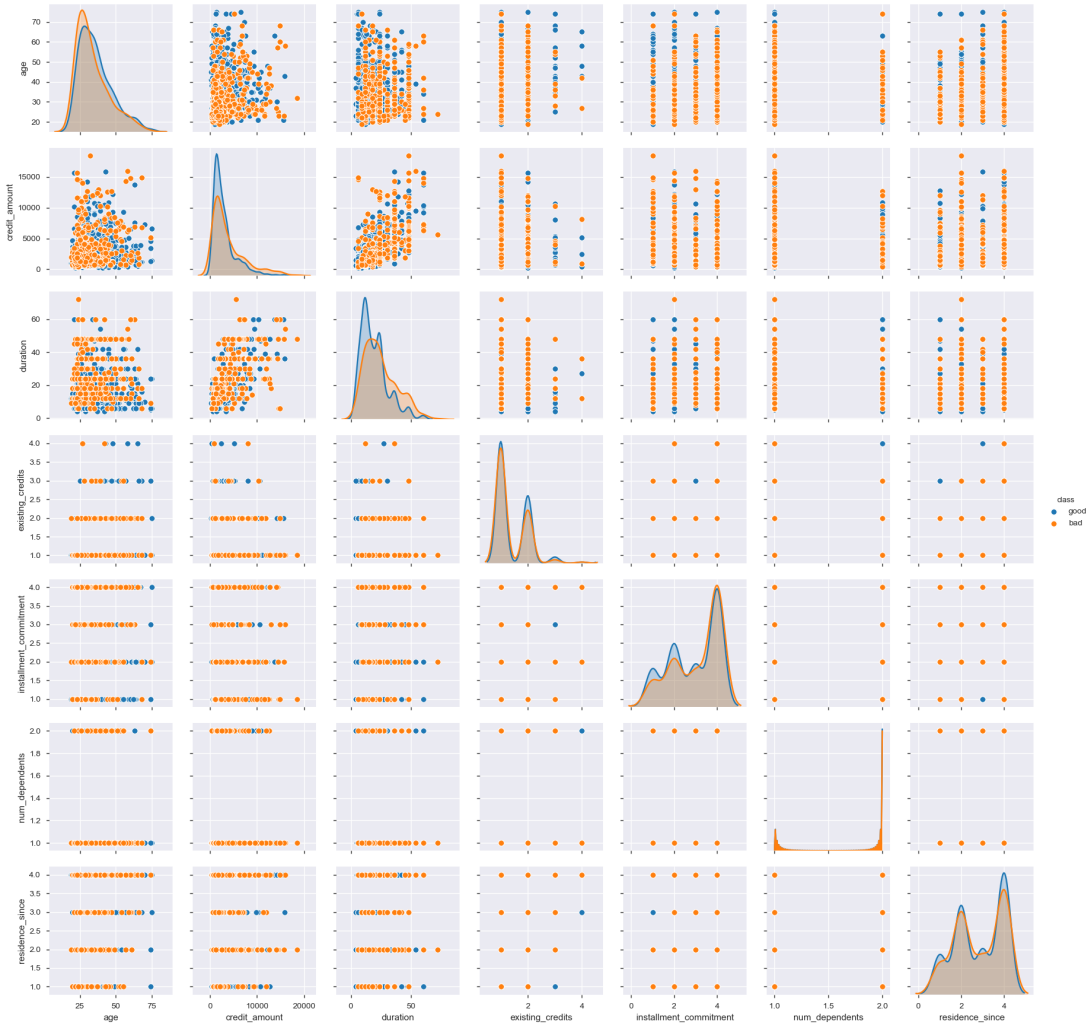
```
In [3]: data_credit_raw = pd.read_csv('dataset_31_credit-g.csv')

In [4]: pandas_profiling.ProfileReport(data_credit_raw)

Out[4]: <pandas_profiling.ProfileReport at 0x7f9d2a0c52b0>
```

2.3 Dividindo valores de atributos

```
In [5]: dict_data = data_credit_raw.to_dict(orient='records')

In [6]: for line in dict_data:
            splited = line['personal_status'].split(sep=' ')
            line['sex'] = splited[0]
            line['personal_status'] = splited[1]

In [7]: data_credit = pd.DataFrame(dict_data)
        aux = data_credit['class']
        data_credit.drop(columns=['class'], inplace = True)
        data_credit['class'] = aux
```

2.4 Visualização

```
In [8]: sns.pairplot(data_credit, diag_kind="kde",hue='class')

Out[8]: <seaborn.axisgrid.PairGrid at 0x7f9d0dc73a20>
```



2.5 Transformação

```
In [9]: data_credit_lb = data_credit.copy()
```

```
In [10]: def list_attrib_categor(data):
             atributos_categoricos = []

             for atributo, tipo in zip(data.columns, data.dtypes):
                 if tipo == object:
                     atributos_categoricos.append(atributo)

             return atributos_categoricos
```

```
In [11]: for atribute in list_attrib_categor(data_credit_lb):
             le = LabelEncoder()
             data_credit_lb[atribute] = le.fit_transform(data_credit[atribute].values)
```

In [12]:
```
data_credit_lb.head()
```

Out[12]:

	age	checking_status	credit_amount	credit_history	duration	employment	\
0	67	1	1169	1	6	3	
1	22	0	5951	3	48	0	
2	49	3	2096	1	12	1	
3	45	1	7882	3	42	1	
4	53	1	4870	2	24	0	

	existing_credits	foreign_worker	housing	installment_commitment	...	\
0	2	1	1	4	...	
1	1	1	1	2	...	
2	1	1	1	2	...	
3	1	1	0	2	...	
4	2	1	0	3	...	

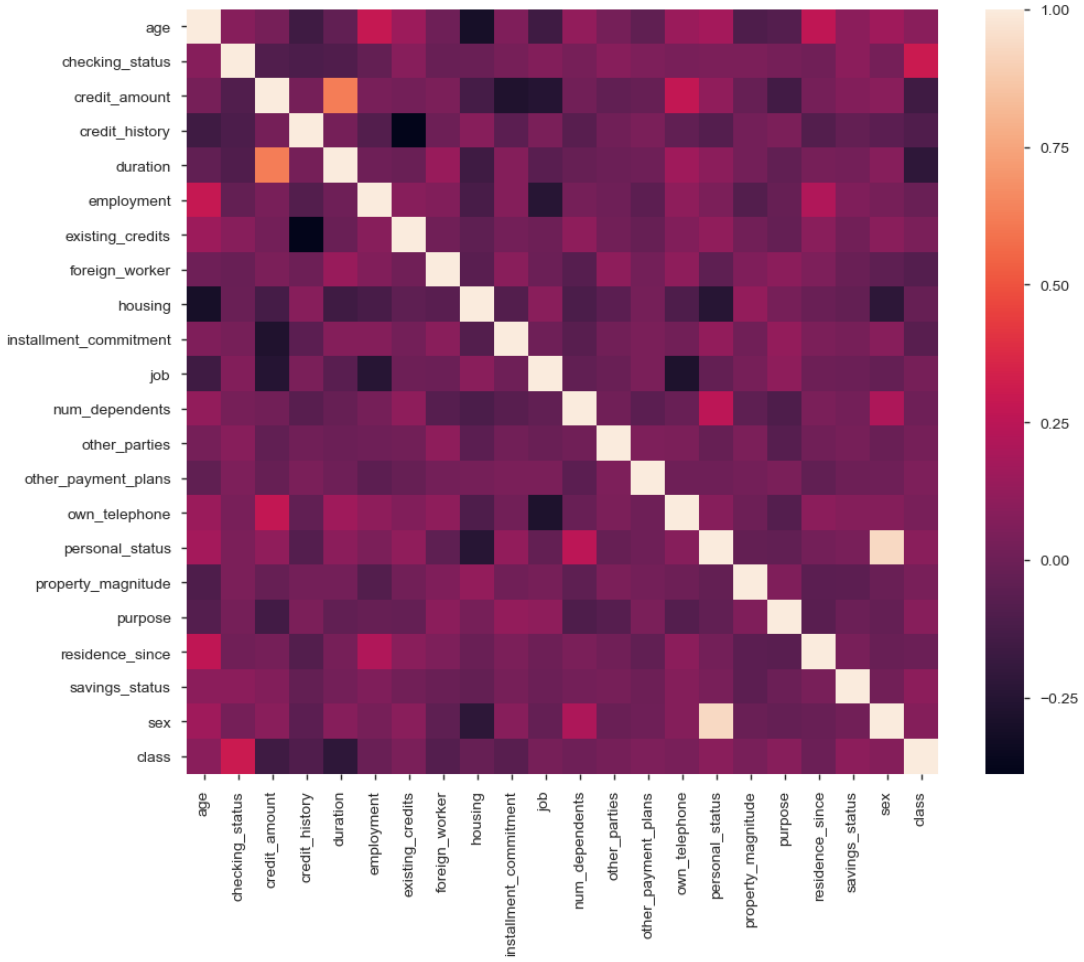
	other_parties	other_payment_plans	own_telephone	personal_status	\
0	2	1	1	3	
1	2	1	0	0	
2	2	1	0	3	
3	1	1	0	3	
4	2	1	0	3	

	property_magnitude	purpose	residence_since	savings_status	sex	class
0	2	7	4	4	1	1
1	2	7	2	2	0	0
2	2	4	3	2	1	1
3	0	5	4	2	1	1
4	1	1	4	2	1	0

[5 rows x 22 columns]

In [13]:
```
plt.subplots(figsize=(11, 9))
sns.heatmap(data_credit_lb.corr())
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9cfa5517f0>




```
In [14]: sns_plot = sns.pairplot(data_credit_lb, diag_kind="kde",hue='class')
         sns_plot.savefig("plot_credit_lbl.png")
```

2.6 Escalonando

```
In [15]: # Data sem escalonar
         data_np_x = data_credit_lb.drop(columns=['class']).to_numpy()
         data_np_y = data_credit_lb['class'].to_numpy()

In [16]: scaler = StandardScaler().fit(data_np_x)
         data_np_x = scaler.transform(data_np_x)
```

2.7 Classificando

2.8 Funções necessárias

```
In [17]: folds_value = 10
```

```
In [18]: def calcula_metricas(metricas, y_test, y_predict):
             metricas['acc'] += (accuracy_score(y_test, y_predict))
             metricas['recall'] += (recall_score(y_test, y_predict))
             metricas['precision'] += (precision_score(y_test, y_predict))
             metricas['f1'] += f1_score(y_test, y_predict)
             metricas['roc'] += roc_auc_score(y_test, y_predict)
             metricas['kappa'] += cohen_kappa_score(y_test, y_predict)
             metricas['balanced_acc'] += balanced_accuracy_score(y_test, y_predict)

In [19]: def save_metricas(name, metricas):
             f = open(name, 'w')
             f.write('Acuária:' + str(metricas['acc']) + '\n')
             f.write('Recall:' + str(metricas['recall']) + '\n')
             f.write('Precisão:' + str(metricas['precision']) + '\n')
             f.write('F-Measure:' + str(metricas['f1']) + '\n')
             f.write('Curva Roc:' + str(metricas['roc']) + '\n')
             f.write('Indice Kappa:' + str(metricas['kappa']) + '\n')
             f.write('Acuária Balanceada:' + str(metricas['balanced_acc']) + '\n')
             f.close()

In [20]: def show_metricas(metricas):
             print('Acuária:', metricas['acc'])
             print('Recall:', metricas['recall'])
             print('Precisão:', metricas['precision'])
             print('F-Measure:', metricas['f1'])
             print('Curva Roc:', metricas['roc'])
             print('Indice Kappa:', metricas['kappa'])
             print('Acuária Balanceada:', metricas['balanced_acc'])

In [21]: def write_metricas(name_file, metricas, metodo):
             f = open(name_file, "a")
             f.write(metodo + ';')
             f.write(str(round(metricas['acc'],4)) + ';')
             f.write(str(round(metricas['recall'],4)) + ';')
             f.write(str(round(metricas['precision'],4)) + ';')
             f.write(str(round(metricas['f1'],4)) + ';')
             f.write(str(round(metricas['roc'],4)) + ';')
             f.write(str(round(metricas['kappa'],4)) + ';')
             f.write(str(round(metricas['balanced_acc'],4)) + '\n')
             f.close()
```

2.9 Aplicando KNN com K-fold

2.10 DataFrame Cru

```
In [22]: formato = 'Cru'

In [23]: folds_value = 10
```

```
In [24]: kf = KFold(n_splits=10, shuffle=True, random_state=random.randint(0, 10))
         data_kfold = kf.split(data_np_x)

         train = []
         test = []

         for train_index, test_index in data_kfold:
             train.append(train_index)
             test.append(test_index)
```

```
In [25]: name_file = 'metricas-' + formato + '.csv'

         f = open(name_file, "w")
         f.write(';Acurácia;Recall;Precisão;F1;Roc;Kappa;Acurácia Balanceada\n')
         f.close()
```

2.11 Aplicando KNN com K-fold

```
In [26]: metodo = 'KNN'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             neigh = KNeighborsClassifier(n_neighbors=1)
             neigh.fit(x_train, y_train)

             y_predict = neigh.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.667
Recall: 0.7861261188753572
Precisão: 0.7484957884013399
F-Measure: 0.7661153571336496
Curva Roc: 0.5875810628023066
Indice Kappa: 0.1792183363179038
Acuária Balanceada: 0.5875810628023066
```

2.12 Aplicando GaussianNB com K-fold

```
In [27]: metodo = 'Gauss'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             gauss = GaussianNB()
             gauss.fit(x_train, y_train)

             y_predict = gauss.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.6890000000000001
Recall: 0.7618699844344405
Precisão: 0.7949540364472436
F-Measure: 0.7729787908572685
Curva Roc: 0.6487859069288866
Indice Kappa: 0.28046520330743563
Acuária Balanceada: 0.6487859069288866
```

2.13 Aplicando DecisionTreeClassifier com K-fold

```
In [28]: metodo = 'Tree'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             tree = DecisionTreeClassifier()
             tree.fit(x_train, y_train)

             y_predict = tree.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10
```

```
        show_metricas(metricas)
        write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.6950000000000001
Recall: 0.771866604583279
Precisão: 0.7854832341718693
F-Measure: 0.7771118530772865
Curva Roc: 0.6412260728515466
Indice Kappa: 0.2811079450129389
Acuária Balanceada: 0.6412260728515466
```

2.14 Aplicando SVM com K-fold

```
In [29]: metodo = 'SVM'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             svm = SVC()
             svm.fit(x_train, y_train)

             y_predict = svm.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7460000000000001
Recall: 0.937453897891975
Precisão: 0.7572620096287589
F-Measure: 0.8369579580637951
Curva Roc: 0.6182334230294194
Indice Kappa: 0.2780091570633585
Acuária Balanceada: 0.6182334230294194
```

2.15 DataFrame Selecionado

2.15.1 Selecionando atributos

```
In [31]: data_select = data_credit_lb[['age','credit_amount','credit_history']]
```

```
In [32]: data_np_x = data_select.to_numpy()
         data_np_y = data_credit_lb['class'].to_numpy()

In [33]: scaler = StandardScaler().fit(data_np_x)
         data_np_x = scaler.transform(data_np_x)
```

2.16 Aplicando

```
In [35]: formato = 'Selecionado'

In [36]: kf = KFold(n_splits=10, shuffle=True, random_state=random.randint(0, 10))
         data_kfold = kf.split(data_np_x)

         train = []
         test = []

         for train_index, test_index in data_kfold:
             train.append(train_index)
             test.append(test_index)

In [37]: name_file = 'metricas-' + formato + '.csv'

         f = open(name_file, "w")
         f.write(';Acurácia;Recall;Precisão;F1;Roc;Kappa;Acurácia Balanceada\n')
         f.close()
```

2.17 Aplicando KNN com K-fold

```
In [38]: metodo = 'KNN'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'bal

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             neigh = KNeighborsClassifier(n_neighbors=1)
             neigh.fit(x_train, y_train)

             y_predict = neigh.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.628
Recall: 0.7340769232336475
Precisão: 0.7351502043587164
F-Measure: 0.7334090073511181
Curva Roc: 0.5578400225857234
Indice Kappa: 0.11274928271974041
Acuária Balanceada: 0.5578400225857234
```

2.18 Aplicando GaussianNB com K-fold

```
In [39]: metodo = 'Gauss'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'bal

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             gauss = GaussianNB()
             gauss.fit(x_train, y_train)

             y_predict = gauss.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.704
Recall: 0.940412783538676
Precisão: 0.7219515928388632
F-Measure: 0.816057832809539
Curva Roc: 0.547566475831035
Indice Kappa: 0.1180194995792841
Acuária Balanceada: 0.547566475831035
```

2.19 Aplicando DecisionTreeClassifier com K-fold

```
In [40]: metodo = 'Tree'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'bal

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]
```

```
            tree = DecisionTreeClassifier()
            tree.fit(x_train, y_train)

            y_predict = tree.predict(x_test)

            calcula_metricas(metricas, y_test, y_predict)

        for metrica, value in metricas.items():
            metricas[metrica] = value/10

        show_metricas(metricas)
        write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.6080000000000001
Recall: 0.7045205059143995
Precisão: 0.7254031765999808
F-Measure: 0.7138764505704455
Curva Roc: 0.5426245796400748
Indice Kappa: 0.08333851389889967
Acuária Balanceada: 0.5426245796400748
```

2.20 Aplicando SVM com K-fold

```
In [41]: metodo = 'SVM'
         metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

         for train_index, test_index in zip(train, test):
             x_train, x_test = data_np_x[train_index], data_np_x[test_index]
             y_train, y_test = data_np_y[train_index], data_np_y[test_index]

             svm = SVC()
             svm.fit(x_train, y_train)

             y_predict = svm.predict(x_test)

             calcula_metricas(metricas, y_test, y_predict)

         for metrica, value in metricas.items():
             metricas[metrica] = value/10

         show_metricas(metricas)
         write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.6940000000000001
Recall: 0.9505694099183885
Precisão: 0.7110855036966002
```

```
F-Measure: 0.8127522669762838
Curva Roc: 0.5237673956555946
Indice Kappa: 0.05857543003277889
Acuária Balanceada: 0.5237673956555946
```

Classificacao2

May 21, 2019

1 0. Introdução

Trabalho:
Aluno: Maicon Dall'Agnol
R.A.: 151161868
Disciplina: Tópico em Aprendizado de Máquina
Objetivos :

- Escolha dois datasets rotulados.
- Realize a análise estatística, visualização e pré-processamento dos dados.
- Realize os experimentos criando duas bases de teste distintas:
- – considerando todos os atributos do dataset;
- – selecionando alguns atributos e descartando outros.
- Aplique três métodos de classificação distintos nas duas bases acima referentes a cada dataset.
- Para cada dataset, em cada uma das bases, analise os resultados segundo medidas de qualidade de classificação, usando índices de validação externa (acurácia, recall, precisão, F-measure, índice Kappa) e cruva ROC.
- Proponha uma maneira adicional de comparar os resultados obtidos além das medidas acima.
- Compare e interprete os resultados dos dois experimentos em cada dataset.
- Faça tabela com as medidas de validação

1.1 0.1 Dependências

Para realização da tarefa foram utilizados as seguintes bibliotecas:

```
In [282]: import pandas as pd
          import numpy as np
          import pandas_profiling

          from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.preprocessing import StandardScaler

# KFold
from sklearn.model_selection import KFold
import random

# Classificadores
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

#Metricas
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from  sklearn.metrics  import  roc_auc_score
from sklearn.metrics import balanced_accuracy_score

#Visualização
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

2 1. Dados

Dataset correspondente a fonemas e composto de atributos abstratos cujo atributo alvo é a classe de som nasais ou orais (classe 1 e 2, respectivamente).

2.1 1.1 Informações sobre os dados:

Atributos:

- V1
- V2
- V2
- V4

Classe:

- Class

2.2 Importando Dataset

```
In [3]: data_phoneme_raw = pd.read_csv('dados/phoneme.csv')
```

```
In [4]: data_phoneme_raw.head()
```

Out[4]:

	V1	V2	V3	V4	V5	Class
0	0.489927	-0.451528	-1.047990	-0.598693	-0.020418	1
1	-0.641265	0.109245	0.292130	-0.916804	0.240223	1
2	0.870593	-0.459862	0.578159	0.806634	0.835248	1
3	-0.628439	-0.316284	1.934295	-1.427099	-0.136583	1
4	-0.596399	0.015938	2.043206	-1.688448	-0.948127	1

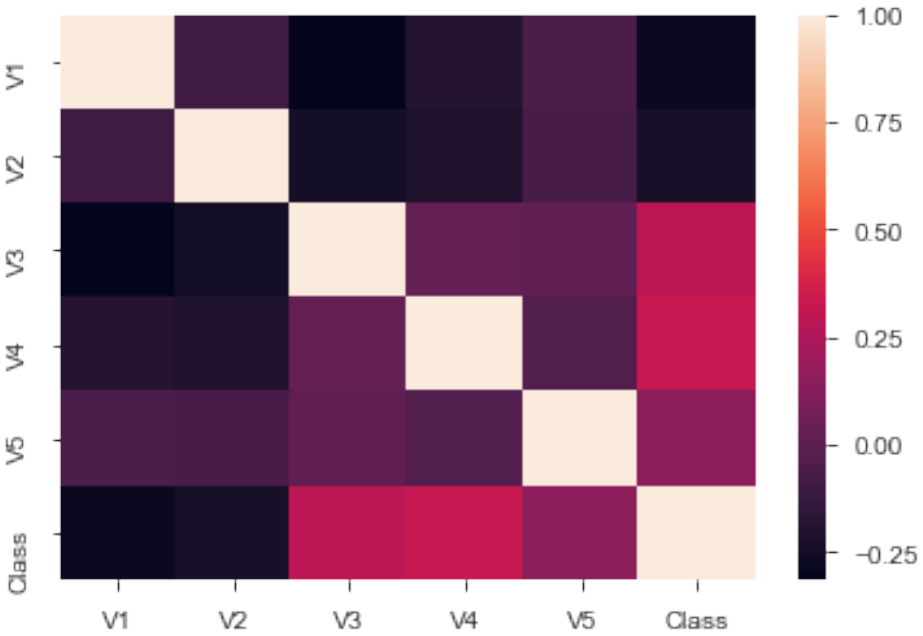
```
In [5]: pandas_profiling.ProfileReport(data_phoneme_raw)
```

```
Out[5]: <pandas_profiling.ProfileReport at 0x7fe097237208>
```

2.3 Visualizações

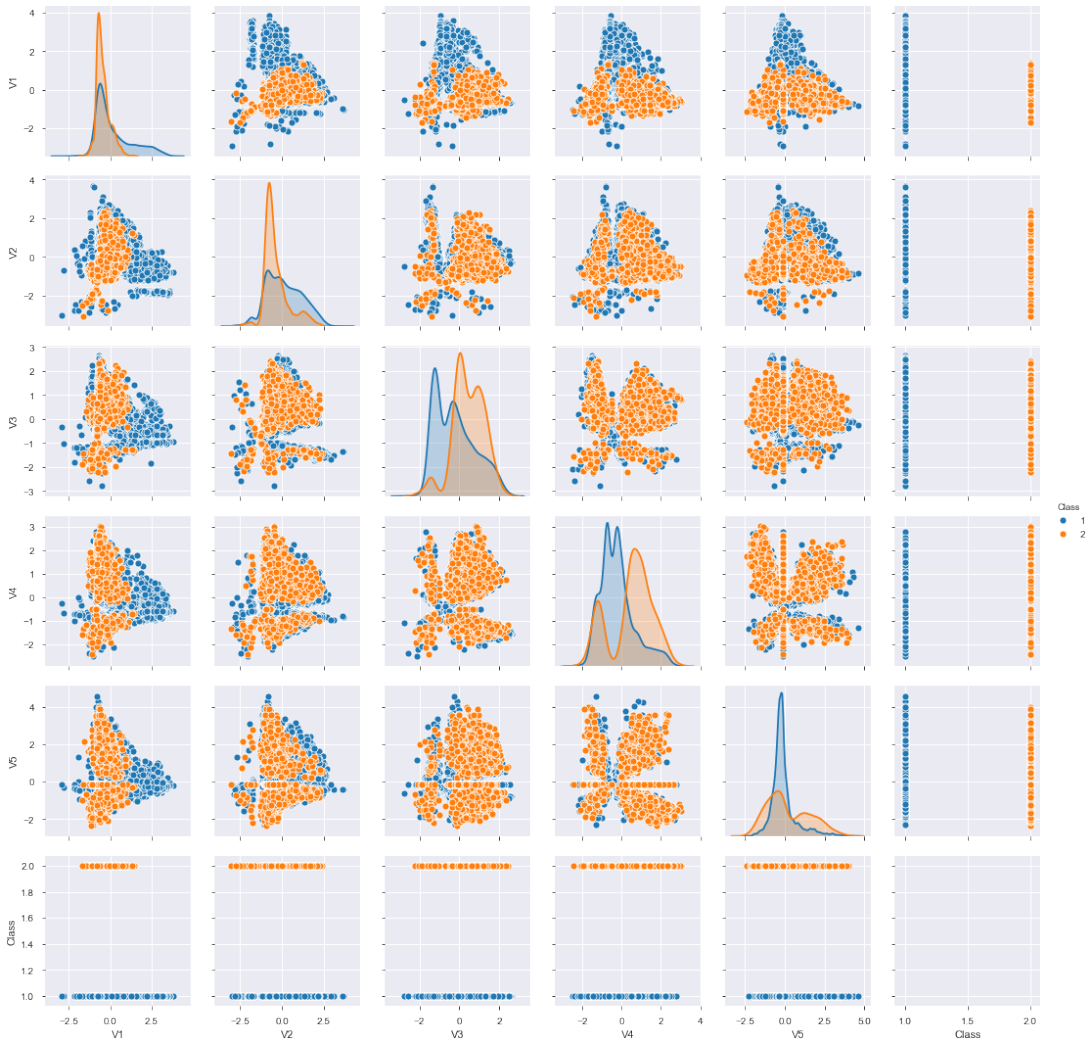
```
In [302]: sns.heatmap(data_phoneme_raw.corr())
```

```
Out[302]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe05351b630>
```



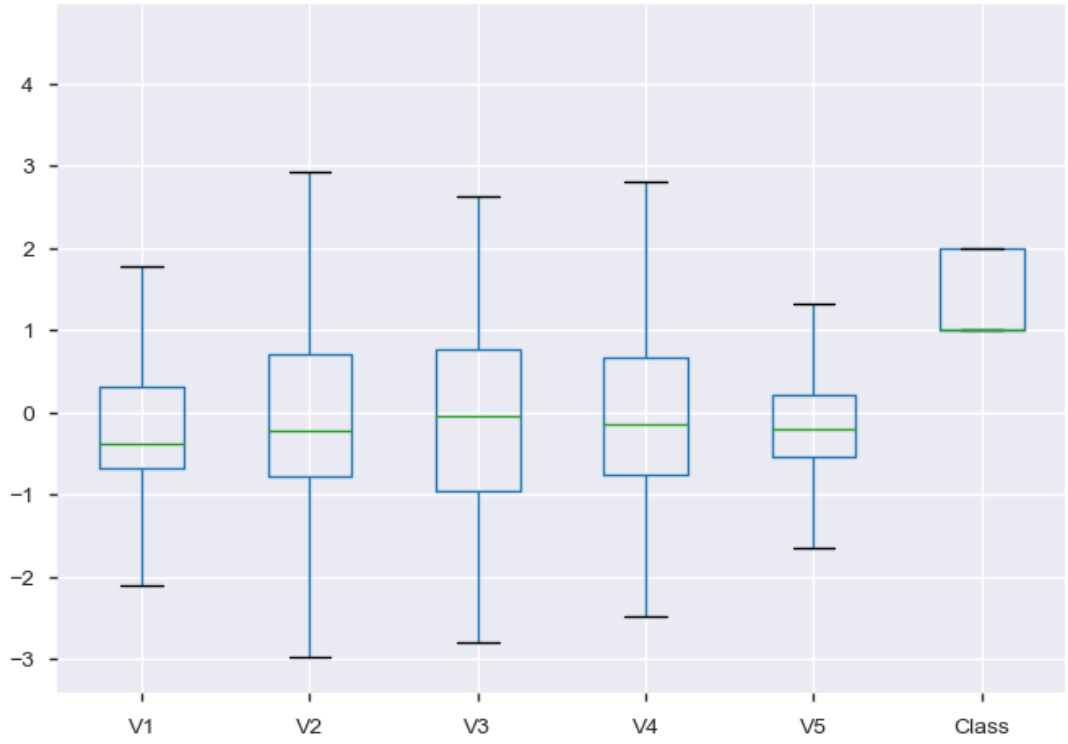

```
In [303]: sns.pairplot(data_phoneme_raw, diag_kind="kde",hue='Class')
```

```
Out[303]: <seaborn.axisgrid.PairGrid at 0x7fe053505208>
```

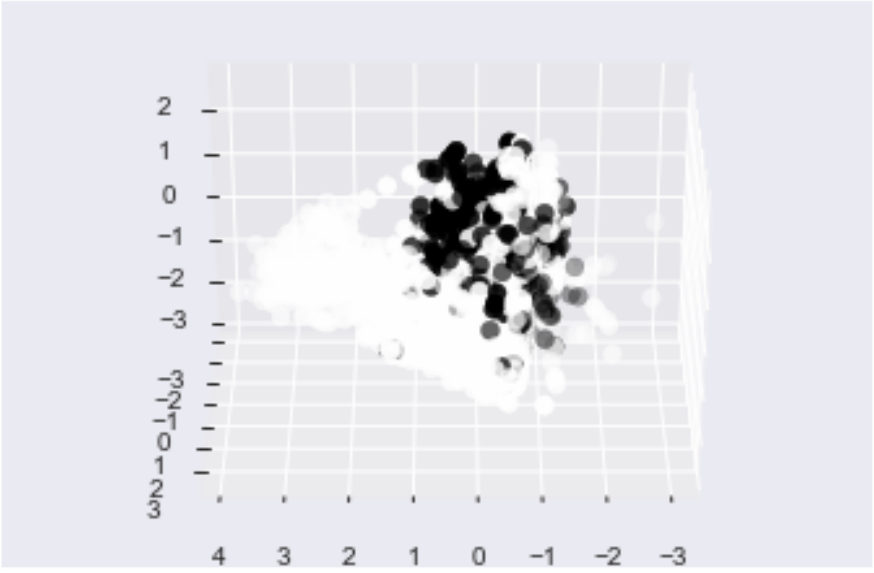


In [103]: `data_phoneme_raw.plot.box()`

Out[103]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fe06670cfd0>`



In [304]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data_phoneme_raw.V1,data_phoneme_raw.V2, data_phoneme_raw.V3, c=data_phon
ax.view_init(30, 90)
plt.show()
```



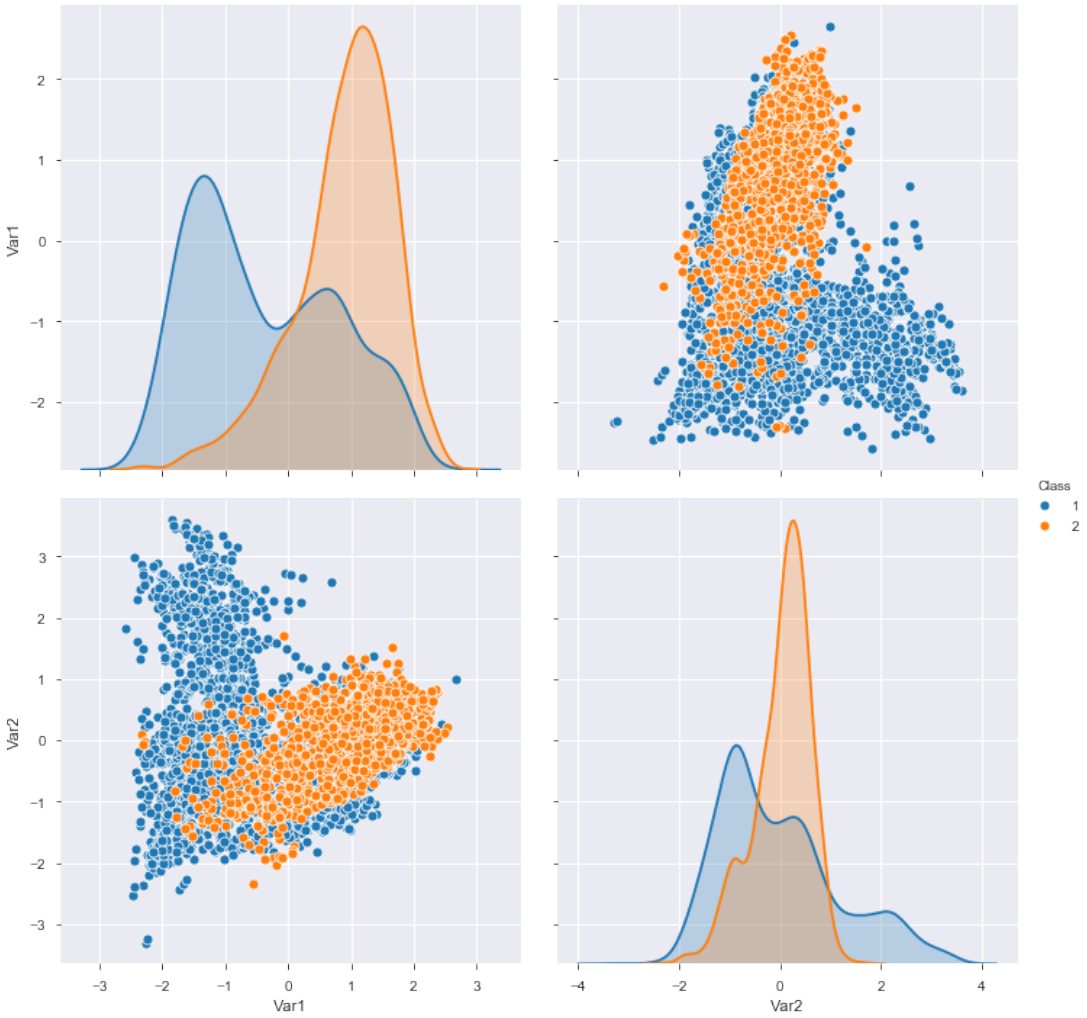
2.4 PCA

```
In [374]: data_pca = PCA(n_components=2).fit_transform(data_phoneme_raw.drop(columns=['Class'])
          data_pca = pd.DataFrame(data_pca, columns = ['Var1','Var2'])
          data_pca['Class'] = data_phoneme_raw['Class']

In [375]: data_pca.head()
```

Out[375]:

	Var1	Var2	Class
0	-0.946325	0.652816	1
1	0.083178	-0.725027	1
2	0.530377	1.037576	1
3	0.990195	-0.796042	1
4	0.705725	-1.142581	1

```
In [376]: sns.pairplot(data_pca, diag_kind="kde", vars = ['Var1', 'Var2'], hue='Class', size =

Out[376]: <seaborn.axisgrid.PairGrid at 0x7fe047e47be0>
```



2.5 Rebalanceando as Classes com Random under-sampling

```
In [272]: count_class_1, count_class_2 = data_phoneme_raw.Class.value_counts()

          class_1_df = data_phoneme_raw[data_phoneme_raw['Class'] == 1]
          class_2_df = data_phoneme_raw[data_phoneme_raw['Class'] == 2]
```

```
In [273]: print('Class1:', count_class_1)
          print('Class2:', count_class_2)
```

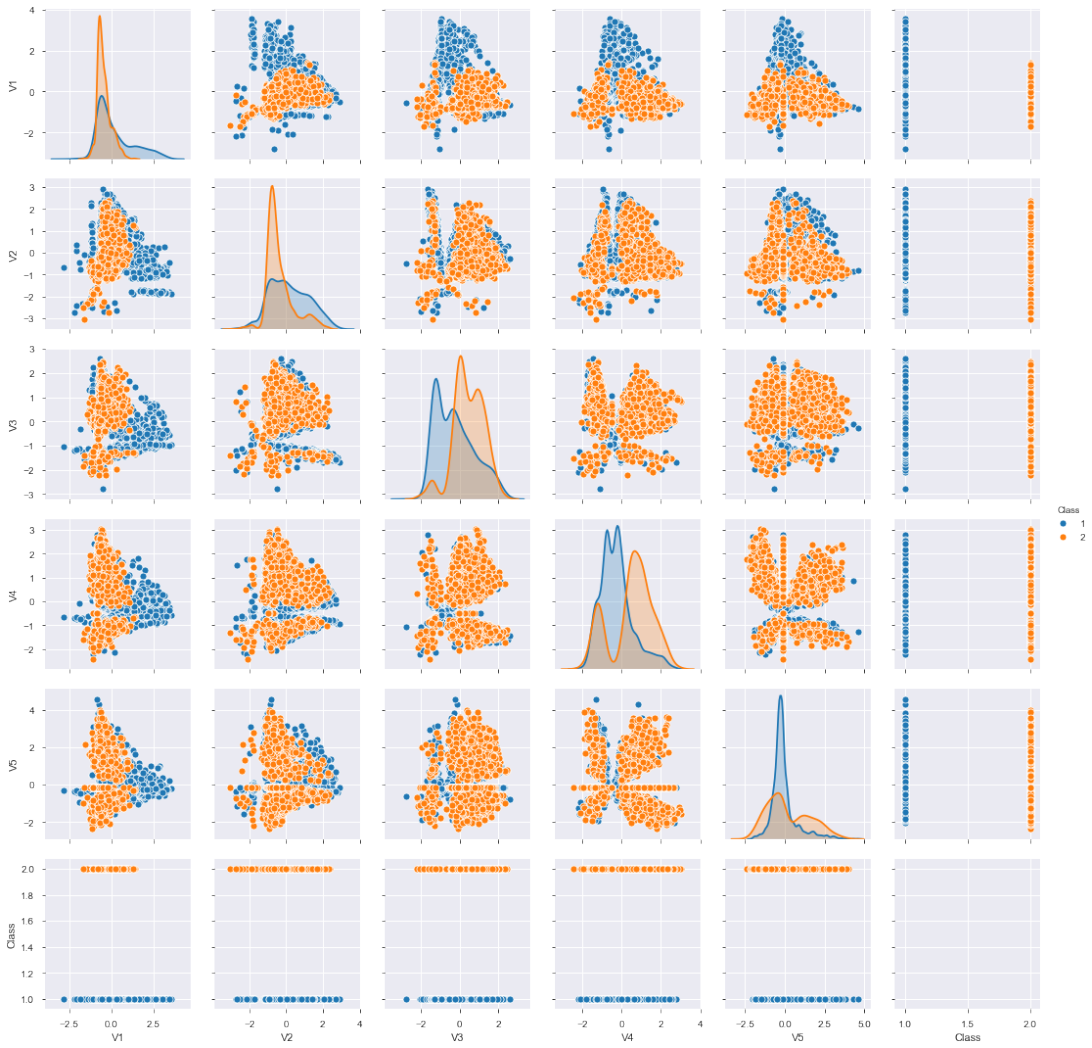
```
Class1: 3818
Class2: 1586
```

```
In [337]: under_class_1_df = class_1_df.sample(count_class_2, random_state=random.randint(2,10)
          balanced_df = pd.concat([under_class_1_df, class_2_df], axis=0)
```

```
print('Random under-sampling:')
print(balanced_df.Class.value_counts())

balanced_df.Class.value_counts().plot(kind='bar', title='Count (target)');
```

```
Random under-sampling:
1    1586
2    1586
Name: Class, dtype: int64
```


```
In [338]: balanced_df.reset_index(inplace=True, drop=True)

In [339]: sns.pairplot(balanced_df, diag_kind="kde",hue='Class')

Out[339]: <seaborn.axisgrid.PairGrid at 0x7fe046103898>
```



2.6 Classificando

2.7 Funções necessárias

```
In [430]: def calcula_metricas(metricas, y_test, y_predict):
              metricas['acc'] += (accuracy_score(y_test, y_predict))
              metricas['recall'] += (recall_score(y_test, y_predict))
              metricas['precision'] += (precision_score(y_test, y_predict))
              metricas['f1'] += f1_score(y_test, y_predict)
              metricas['roc'] += roc_auc_score(y_test, y_predict)
              metricas['kappa'] += cohen_kappa_score(y_test, y_predict)
              metricas['balanced_acc'] += balanced_accuracy_score(y_test, y_predict)
```

```
In [431]: def save_metricas(name, metricas):
              f = open(name, 'w')
```

```
            f.write('Acuária:' + str(metricas['acc']) + '\n')
            f.write('Recall:' + str(metricas['recall']) + '\n')
            f.write('Precisão:' + str(metricas['precision']) + '\n')
            f.write('F-Measure:' + str(metricas['f1']) + '\n')
            f.write('Curva Roc:' + str(metricas['roc']) + '\n')
            f.write('Indice Kappa:' + str(metricas['kappa']) + '\n')
            f.write('Acuária Balanceada:' + str(metricas['balanced_acc']) + '\n')
            f.close()
```

```
In [432]: def show_metricas(metricas):
              print('Acuária:', metricas['acc'])
              print('Recall:', metricas['recall'])
              print('Precisão:', metricas['precision'])
              print('F-Measure:', metricas['f1'])
              print('Curva Roc:', metricas['roc'])
              print('Indice Kappa:', metricas['kappa'])
              print('Acuária Balanceada:', metricas['balanced_acc'])
```

```
In [433]: def write_metricas(name_file, metricas, metodo):
              f = open(name_file, "a")
              f.write(metodo + ';')
              f.write(str(round(metricas['acc'],4)) + ';')
              f.write(str(round(metricas['recall'],4)) + ';')
              f.write(str(round(metricas['precision'],4)) + ';')
              f.write(str(round(metricas['f1'],4)) + ';')
              f.write(str(round(metricas['roc'],4)) + ';')
              f.write(str(round(metricas['kappa'],4)) + ';')
              f.write(str(round(metricas['balanced_acc'],4)) + '\n')
              f.close()
```

2.8 DataFrame Cru

```
In [456]: formato = 'Cru'
```

```
In [457]: data_np_x = data_phoneme_raw.drop_duplicates().drop(columns=['Class']).to_numpy()
          data_np_y = data_phoneme_raw.drop_duplicates()['Class'].to_numpy()
```

```
In [458]: folds_value = 10
```

```
In [459]: kf = KFold(n_splits=10, shuffle=True, random_state=random.randint(0, 10))
          data_kfold = kf.split(data_np_x)

          train = []
          test = []

          for train_index, test_index in data_kfold:
              train.append(train_index)
              test.append(test_index)
```

```
In [460]: name_file = 'metricas-' + formato + '.csv'

          f = open(name_file, "w")
          f.write(';Acurácia;Recall;Precisão;F1;Roc;Kappa;Acurácia Balanceada\n')
          f.close()
```

2.9 Aplicando KNN com K-fold

```
In [461]: metodo = 'KNN'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              neigh = KNeighborsClassifier(n_neighbors=1)
              neigh.fit(x_train, y_train)

              y_predict = neigh.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.9032357589500448
Recall: 0.9448511775439951
Precisão: 0.9204319683648994
F-Measure: 0.932438221179118
Curva Roc: 0.8734025919392348
Indice Kappa: 0.7612078035934119
Acuária Balanceada: 0.8734025919392346
```

2.10 Aplicando GaussianNB com K-fold

```
In [462]: metodo = 'Gauss'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              gauss = GaussianNB()
              gauss.fit(x_train, y_train)
```

```
            y_predict = gauss.predict(x_test)

            calcula_metricas(metricas, y_test, y_predict)

        for metrica, value in metricas.items():
            metricas[metrica] = value/10

        show_metricas(metricas)
        write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7601587301587303
Recall: 0.7734891686424598
Precisão: 0.8730532145319696
F-Measure: 0.819959951920828
Curva Roc: 0.7502026641758153
Indice Kappa: 0.4635814301249811
Acuária Balanceada: 0.7502026641758153
```

2.11 Aplicando DecisionTreeClassifier com K-fold

```
In [463]: metodo = 'Tree'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              tree = DecisionTreeClassifier()
              tree.fit(x_train, y_train)

              y_predict = tree.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.8711784511784512
Recall: 0.9108228302643171
Precisão: 0.9072069404161821
F-Measure: 0.9089766766727794
Curva Roc: 0.8431826126971625
Indice Kappa: 0.6879177531835117
```

```
Acuária Balanceada: 0.8431826126971625
```

2.12 Aplicando SVM com K-fold

```
In [464]: metodo = 'SVM'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              svm = SVC()
              svm.fit(x_train, y_train)

              y_predict = svm.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.8487487116058544
Recall: 0.896382041737635
Precisão: 0.8903387040937798
F-Measure: 0.8932707817698
Curva Roc: 0.814812352136839
Indice Kappa: 0.6326600518433374
Acuária Balanceada: 0.8148123521368389
```

2.13 DataFrame PCA

```
In [465]: formato = 'PCA'
```

```
In [466]: data_np_x = data_pca.drop_duplicates().drop(columns=['Class']).to_numpy()
          data_np_y = data_pca.drop_duplicates()['Class'].to_numpy()
```

```
In [467]: folds_value = 10
```

```
In [468]: kf = KFold(n_splits=10, shuffle=True, random_state=random.randint(0, 10))
          data_kfold = kf.split(data_np_x)

          train = []
          test = []
```

```
            for train_index, test_index in data_kfold:
                train.append(train_index)
                test.append(test_index)
```

```
In [469]: name_file = 'metricas-' + formato + '.csv'

            f = open(name_file, "w")
            f.write(';Acurácia;Recall;Precisão;F1;Roc;Kappa;Acurácia Balanceada\n')
            f.close()
```

2.14 Aplicando KNN com K-fold

```
In [470]: metodo = 'KNN'
            metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

            for train_index, test_index in zip(train, test):
                x_train, x_test = data_np_x[train_index], data_np_x[test_index]
                y_train, y_test = data_np_y[train_index], data_np_y[test_index]

                neigh = KNeighborsClassifier(n_neighbors=1)
                neigh.fit(x_train, y_train)

                y_predict = neigh.predict(x_test)

                calcula_metricas(metricas, y_test, y_predict)

            for metrica, value in metricas.items():
                metricas[metrica] = value/10

            show_metricas(metricas)
            write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7801735037449323
Recall: 0.8450212951774543
Precisão: 0.8441342541984499
F-Measure: 0.8445076543254452
Curva Roc: 0.7338001421595584
Indice Kappa: 0.46787364047429725
Acuária Balanceada: 0.7338001421595584
```

2.15 Aplicando GaussianNB com K-fold

```
In [471]: metodo = 'Gauss'
            metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

            for train_index, test_index in zip(train, test):
                x_train, x_test = data_np_x[train_index], data_np_x[test_index]
```

```
            y_train, y_test = data_np_y[train_index], data_np_y[test_index]

            gauss = GaussianNB()
            gauss.fit(x_train, y_train)

            y_predict = gauss.predict(x_test)

            calcula_metricas(metricas, y_test, y_predict)

        for metrica, value in metricas.items():
            metricas[metrica] = value/10

        show_metricas(metricas)
        write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7792475778190063
Recall: 0.7870410829301134
Precisão: 0.8881304661780701
F-Measure: 0.8344378906510175
Curva Roc: 0.7731816613604637
Indice Kappa: 0.5057835947623386
Acuária Balanceada: 0.7731816613604637
```

2.16 Aplicando DecisionTreeClassifier com K-fold

```
In [472]: metodo = 'Tree'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              tree = DecisionTreeClassifier()
              tree.fit(x_train, y_train)

              y_predict = tree.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7755370026798598
Recall: 0.8364393341690827
```

```
Precisão: 0.8448544863634206
F-Measure: 0.840492706647386
Curva Roc: 0.7318464461247458
Indice Kappa: 0.4605835521962433
Acuária Balanceada: 0.7318464461247458
```

2.17 Aplicando SVM com K-fold

```
In [473]: metodo = 'SVM'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              svm = SVC()
              svm.fit(x_train, y_train)

              y_predict = svm.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7951865594722738
Recall: 0.8322422854221699
Precisão: 0.8722890167173055
F-Measure: 0.8516504561170238
Curva Roc: 0.7682771005851456
Indice Kappa: 0.5199464856631275
Acuária Balanceada: 0.7682771005851456
```

2.18 DataFrame Balanceado

```
In [474]: formato = 'Balanceado'

In [475]: data_np_x = balanced_df.drop_duplicates().drop(columns=['Class']).to_numpy()
          data_np_y = balanced_df.drop_duplicates()['Class'].to_numpy()

In [476]: folds_value = 10

In [477]: kf = KFold(n_splits=10, shuffle=True, random_state=random.randint(0, 10))
          data_kfold = kf.split(data_np_x)
```

```
train = []
test = []

for train_index, test_index in data_kfold:
    train.append(train_index)
    test.append(test_index)
```

```
In [478]: name_file = 'metricas-' + formato + '.csv'

f = open(name_file, "w")
f.write(';Acurácia;Recall;Precisão;F1;Roc;Kappa;Acurácia Balanceada\n')
f.close()
```

2.19 Aplicando KNN com K-fold

```
In [479]: metodo = 'KNN'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

for train_index, test_index in zip(train, test):
    x_train, x_test = data_np_x[train_index], data_np_x[test_index]
    y_train, y_test = data_np_y[train_index], data_np_y[test_index]

    neigh = KNeighborsClassifier(n_neighbors=1)
    neigh.fit(x_train, y_train)

    y_predict = neigh.predict(x_test)

    calcula_metricas(metricas, y_test, y_predict)

for metrica, value in metricas.items():
    metricas[metrica] = value/10

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.8700555045322046
Recall: 0.8684301031357075
Precisão: 0.8720693245828419
F-Measure: 0.8699248356969236
Curva Roc: 0.8701609387223972
Indice Kappa: 0.7395285925502113
Acuária Balanceada: 0.8701609387223972
```

2.20 Aplicando GaussianNB com K-fold

```
In [480]: metodo = 'Gauss'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba
```

```
        for train_index, test_index in zip(train, test):
            x_train, x_test = data_np_x[train_index], data_np_x[test_index]
            y_train, y_test = data_np_y[train_index], data_np_y[test_index]

            gauss = GaussianNB()
            gauss.fit(x_train, y_train)

            y_predict = gauss.predict(x_test)

            calcula_metricas(metricas, y_test, y_predict)

        for metrica, value in metricas.items():
            metricas[metrica] = value/10

        show_metricas(metricas)
        write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.7442069640218825
Recall: 0.6631589812319786
Precisão: 0.7924692452982228
F-Measure: 0.7212104770648144
Curva Roc: 0.7440491041047028
Indice Kappa: 0.4874987636843975
Acuária Balanceada: 0.7440491041047028
```

2.21 Aplicando DecisionTreeClassifier com K-fold

```
In [481]: metodo = 'Tree'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              tree = DecisionTreeClassifier()
              tree.fit(x_train, y_train)

              y_predict = tree.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.8365431457892425
Recall: 0.8388445212265723
Precisão: 0.837190578980462
F-Measure: 0.8373937923510791
Curva Roc: 0.8363343132041047
Indice Kappa: 0.6721658819534905
Acuária Balanceada: 0.8363343132041047
```

2.22 Aplicando SVM com K-fold

```
In [482]: metodo = 'SVM'
          metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa': 0, 'ba

          for train_index, test_index in zip(train, test):
              x_train, x_test = data_np_x[train_index], data_np_x[test_index]
              y_train, y_test = data_np_y[train_index], data_np_y[test_index]

              svm = SVC()
              svm.fit(x_train, y_train)

              y_predict = svm.predict(x_test)

              calcula_metricas(metricas, y_test, y_predict)

          for metrica, value in metricas.items():
              metricas[metrica] = value/10

          show_metricas(metricas)
          write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.8438046559916945
Recall: 0.7710744526685108
Precisão: 0.9033264871646102
F-Measure: 0.8310090726922817
Curva Roc: 0.8440004071747682
Indice Kappa: 0.6870582735808017
Acuária Balanceada: 0.844000407174768
```

Referências

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 2.